МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Отчет по Лабораторной работе**

**Обратная польская запись**

**Выполнил:** студент группы 381808-2

Алёхин Денис Андреевич

Нижний Новгород

2019

Содержание

[1. Введение 3](#_Toc24754091)

[2. Постановка задачи 4](#_Toc24754092)

[3. Руководство пользователя 5](#_Toc24754093)

[4. Руководство программиста 6](#_Toc24754094)

[4.1 Описание структуры программы 6](#_Toc24754095)

[4.2 Описание структур данных 7](#_Toc24754096)

[4.3 Описание алгоритмов 8](#_Toc24754097)

[5. Заключение 9](#_Toc24754098)

[Приложение: Фрагменты исходного кода 10](#_Toc24754099)

[ReversePolishNotation.h 10](#_Toc24754100)

[ReversePolishNotation.cpp 10](#_Toc24754101)

# Введение

В математике существует древняя традиция помещать оператор между операндами (x+y), а не после операндов (xy+). Форма с оператором между операндами называется инфиксной записью.

Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Во-первых, любая формула может быть выражена без скобок. Во-вторых, она [удобна для вычисления формул в машинах со стеками](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C#.D0.92.D1.8B.D1.87.D0.B8.D1.81.D0.BB.D0.B5.D0.BD.D0.B8.D1.8F_.D0.BD.D0.B0_.D1.81.D1.82.D0.B5.D0.BA.D0.B5). В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

# Постановка задачи

Цель работы – разработка программы, позволяющей переводить выражения из префиксного вида в постфиксный, способной подсчитать результат.

Выполнение работы предполагает решение следующих задач:

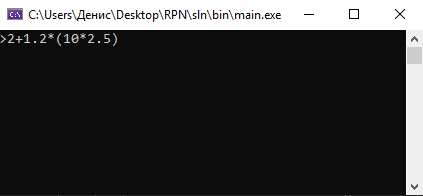
1. Разработка интерфейса класса Postfix.
2. Реализация методов класса Postfix.
3. Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

В ходе выполнения работы был получена библиотека ReversePolishNotation.

Пример использования данной библиотеки:

1. Программа просит ввести выражения для вычисления:



1. Программа выводит представление данного выражения в обратной польской записи и результат вычисления, после чего опять просит ввести выражение:



# Руководство программиста

## 4.1 Описание структуры программы

* docs — инструкции по выполнению лабораторной работы, полезные документы.
* include — директория для размещения заголовочных файлов.
* samples — директория для размещения демо-приложений.
* sln — директория с файлами решений и проектов для VS 2008 и VS 2010, вложенные директории vc9 и vc10 соответственно.
* src — директория с исходными кодами (cpp-файлы).
* README.md — информация о проекте, которую вы сейчас читаете.
* Служебные файлы
  + .gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
  + CMakeLists.txt — корневой файл для сборки проекта с помощью CMake. Может быть использован для генерации проекта в среде разработки, отличной от Microsoft Visual Studio.

## Описание структур данных

Класс Postfix содержит два поля: infix и postfix, содержащие инфиксную и постфиксную запись соответственно.

class Postfix

{

public:

Postfix() : infix(""), postfix("") {}

Postfix(const string expr ) : infix(END + expr + END), postfix("") {} // infix = |expr|

string getInfix() { return infix; }

string getPostfix() { return postfix; }

void convert(); // transformation infix to postfix

double calculate(); //calculation of a expression value

private:

string infix;

string postfix;

void parse(const string, vector<string>&); // first argument is string to parse and second where to parse

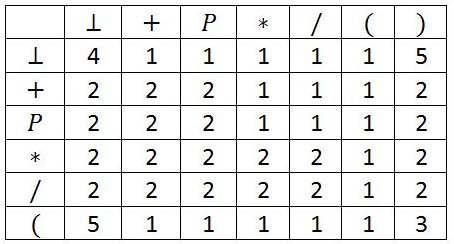
int priority(char, char); // it return value, that says should symbol go: stack or postfix string

};

## Описание алгоритмов

Программа состоит из четырех главных функций:

1. Priority – определяет распределение операций в стек и в постфиксную запись в соответствии с данной таблицей:



Где:

1. Операция отправляется в стек
2. Операция из стека отправляется обратную польскую запись
3. Удаляется операция поступившая на вход и операция из стека
4. Конец программы
5. Сбой
6. Pasre – разделяется строку на лексемы
7. Convert – преобразую инфиксную запись в постфиксную
8. Calculate – вычисляет значение выражения

# Заключение

В ходе лабораторной работы был получен класс на основе которого была создана библиотека, позволяющая переводить выражение из префиксной записи в постфиксную и выводить результат выражения.

# Приложение: Фрагменты исходного кода

## ReversePolishNotation.h

#pragma once

#include <string>

#include <vector>

using namespace std;

const char END = '|';

class Postfix

{

public:

Postfix() : infix(""), postfix("") {}

Postfix(const string expr ) : infix(END + expr + END), postfix("") {} // infix = |expr|

string getInfix() { return infix; }

string getPostfix() { return postfix; }

void convert(); // transformation infix to postfix

double calculate(); //calculation of a expression value

private:

string infix;

string postfix;

void parse(const string, vector<string>&); // first argument is string to parse and second where to parse

int priority(char, char); // it return value, that says should symbol go: stack or postfix string

};

## ReversePolishNotation.cpp

#include "ReversePolishNotatin.h"

#include <stack>

#include <iostream>

bool notUnary(char symb)

{

if (symb == '\*' || symb == '/' || symb == '|' || symb == '(' || symb == ')') return true;

return false;

}

bool isnumber(string numb)

{

if (numb == "+" || numb == "-") return false;

for (int i = 0; i < numb.length(); i++)

if (!(numb[i] == '.' || numb[i] == '+' || numb[i] == '-' || isdigit(numb[i])))

return false;

return true;

}

// Return values from priority function:

//

// | + - \* / ( )

// | 4 1 1 1 1 1 5

// + 2 1 1 1 1 1 2

// - 2 2 2 1 1 1 2

// \* 2 2 2 2 2 1 2

// / 2 2 2 2 2 1 2

// ( 5 1 1 1 1 1 3

int Postfix::priority(char stacksymb, char insymb)

{

switch (insymb)

{

case('|'):

switch (stacksymb)

{

case('|'):

return 4;

case('+'): case('-'): case('\*'): case('/'):

return 2;

case('('): default:

return 5;

}

case('+'): case('-'):

switch (stacksymb)

{

case('|'): case('('):

return 1;

case('+'): case('-'):

return 1;

case('\*'): case('/'):

return 2;

default:

return 5;

}

case('\*'): case('/'):

switch (stacksymb)

{

case('|'): case('+'): case('-'): case('('):

return 1;

case('\*'): case('/'):

return 2;

default:

return 5;

}

case('('):

return 1;

case(')'):

switch (stacksymb)

{

case('|'):

return 5;

case('+'): case('-'): case('\*'): case('/'):

return 2;

case('('):

return 3;

default:

return 5;

}

default:

return 5;

}

}

void unaryTransf(string& number) // --n -> n; ++n -> n

{

bool minus = false;

string newNumb = "";

for (int i = 0; i < number.length(); i++)

{

if (number[i] == '-')

{

if (minus) minus = false;

else minus = true;

}

if (isdigit(number[i]) || number[i] == '.') newNumb += number[i];

}

number = "";

if (minus == true)

number = '-' + newNumb;

else number = newNumb;

}

void Postfix::parse(const string str, vector<string>& data)

{

string number = "";

for (int i = 0; i < str.length(); i++)

{

while (str[i] != ' ' && !notUnary(str[i]) && i != str.length()-1) //reading a number

{

// if + or - isn't unary

if ((str[i] == '+' || str[i] == '-') && (i > 0) && (isdigit(str[i - 1]))) break;

number += str[i];

i++;

}

if (!isnumber(number)) // the way, when in number is + or -

{

data.push\_back(number);

number = "";

}

if (number != "")

{

unaryTransf(number); // --n -> n; ++n -> n

try { stod(number); } // checking to the correct number

catch (exception & e)

{

throw("Wrong input");

}

data.push\_back(number);

number = "";

}

if (str[i] == ' ')

{

continue;

number = "";

}

else

{

string s(1, str[i]); // to convert char to string

data.push\_back(s);

}

}

}

void Postfix::convert()

{

vector<string> data;

parse(infix, data);

stack<char> tokens;

tokens.push(END); // start symbol

for (int i = 1; i < data.size(); i++)

{

if (isnumber(data[i]))

postfix += data[i];

else

{

postfix += " ";

int type = priority(tokens.top(), data[i][0]);

switch (type)

{

case(1): // 1 - put insymb to stack line

tokens.push(data[i][0]);

break;

case(2): // 2 - take symbol from stack and put it to postfix line

postfix += tokens.top();

tokens.pop();

i--;

continue;

case(3): // 3 - delete insymb and stacksymb

tokens.pop();

continue;

break;

case(4): // 4 - succsessful end

return;

case(5): default: // 5 - wrong input

throw ("\nWrong input");

break;

}

}

}

}

double Postfix::calculate()

{

vector<string> data;

parse(postfix, data);

stack<double> numbers;

double number = 0;

for (int i = 0; i < data.size(); i++)

{

if (isnumber(data[i]))

numbers.push(stod(data[i]));

else

{

double temp;

if (numbers.empty()) throw("Wrong input");

temp = numbers.top();

numbers.pop();

if(numbers.empty()) throw("Wrong input");

switch (data[i][0])

{

case('+'):

temp = numbers.top() + temp;

break;

case('-'):

temp = numbers.top() - temp;

break;

case('\*'):

temp = numbers.top() \* temp;

break;

case('/'):

temp = numbers.top() / temp;

break;

default:

throw("Wrong input");

}

numbers.pop();

numbers.push(temp);

}

}

if (numbers.size() > 1) throw("Wrong input");

return numbers.top();

}